Convolutional Neural Networks vs Graph Neural Networks for Image Classification

Syd Proom (N0788203), Supervised by Dr. Archontis Giannakidis

Department of Physics & Mathematics Nottingham Trent University

INTRODUCTION

- Image classification is used in many applications and industries.
- Machine learning is a growing area of research which aims to improve classification and regression problems by allowing computers to learn complex features in datasets.
- Neural networks are a subset of machine learning which seek to emulate how the human brain predicts things.
- Typically Convolutional Neural Networks (CNN) are trained for image predictions due to their ability to deal with large input data with relatively little compute power required.
- By converting input image data into graphs, Graph Neural Networks (GNNs) can be used on image datasets.
- Both CNNs and GNNs have multiple architectures to choose from depending on the dataset, along with individual hyperparameters which need to be tuned to get acceptable results.
- This poster will aim to compare the performance of CNNs and GNNs for image classification on a standard dataset called MNIST which features thousands of hand-written digits to identify.

NEURAL NETWORK THEORY

- Neurons in a Neural Network are connected in layers with a threshold which is adjusted depending on the data. There is typically an input layer, multiple hidden layers and an output layer which depends on the number of classes which are shown in Figure 1.

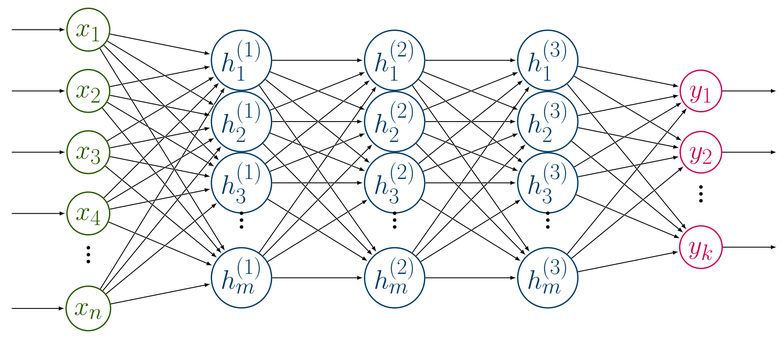

Figure 1: General neural network structure with input (green), hidden (blue) and output (pink) layers.

- Each neurons activation z is determined by all previous layer neurons using a weighted sum of the input x_i and weights w_i which is tweaked using a bias term b to reduce overfitting to the dataset:

$$z = w_1x_1 + w_2x_2 + ... + w_nx_n + b = \sum_1^n w_ix_i + b = \boldsymbol{x}^T\boldsymbol{w} + b$$

CNN ARCHITECTURE

- CNNs are used on multi-dimensional input data (2D images, 3D scans) as normal neural networks do not scale efficiently to high dimension data.
- CNNs reduce the size of input data using convolutional layers in Figure 2 which has adjustable size kernel and stride (how much the kernel moves):

$$\begin{array}{|c|c|c|c|c|}\hline 1&2&5&2&1\\\hline 0&6&4&4&1\\\hline 0&3&4&3&2\\\hline 4&2&2&6&3\\\hline 1&0&4&5&3\\\hline\end{array} \bullet \begin{array}{|c|c|}\hline 1&2\\\hline 3&4\\\hline\end{array} = \begin{array}{|c|c|c|c|}\hline 29&46&37&20\\\hline 24&39&36&23\\\hline 26&25&40&36\\\hline 11&22&46&39\\\hline\end{array}$$

Figure 2: Example of a convolution layer acting on an (5×5) input array (left) with a (2×2) kernel (center) and output (4×4) feature map (right) using a (1×1) stride.

- Another way to reduce complexity is using pooling layers as seen in Figure 3, which are typically used after convolution layers.

$$\begin{array}{|c|c|c|}\hline 0&3&4\\\hline 4&2&2\\\hline 1&0&4\\\hline\end{array} \longrightarrow \begin{array}{|c|c|}\hline 4&4\\\hline 4&4\\\hline\end{array} \qquad \begin{array}{|c|c|c|}\hline 0&3&4\\\hline 4&2&2\\\hline 1&0&4\\\hline\end{array} \longrightarrow \begin{array}{|c|c|}\hline 2.25&2.75\\\hline 1.75&2\\\hline\end{array}$$

(a) (b)

Figure 3: (a) shows max pooling on a (3×3) input with a (2×2) pool size and (1×1) stride resulting in a (2×2) output. (b) shows average pooling on a (3×3) input with a (2×2) pool size and (1×1) stride resulting in a (2×2) output.

- Once the data has been reduced in complexity using convolution, it goes through a network similar to that seen in Figure 1 by flattening the 2D data into a 1D vector.
- The activation of each neuron is tweaked to minimise a loss function ℓ in the equation below by differentiating using an optimiser which both can be chosen and calculated using backpropagation over the size of the batch N:

$$\ell_i(y_i, p_i) = -N \sum_{i=1}^{N} y_i \log(p_i)$$

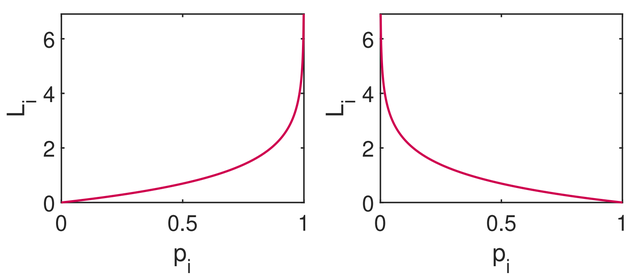

Figure 4: Two graphs showing loss function with the left predicting label with actual label "$y_i = 0$" and the right predicting label with actual label "$y_i = 1$" with both predicted label p_i.

GNN ARCHITECTURE

- GNNs require graph data, which includes time series data such as speech, molecules or social networks [7].
- Images can be converted to graphs using "superpixels" as shown in Figure 5.

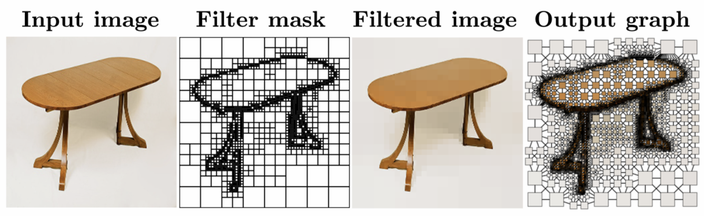

Figure 5: Example of conversion process from image to graph for classification [6].

- Superpixels are useful as normal pixels in an image are not all as important as eachother, which superpixels solves by emphasising useful information and reducing storage dedicated to less useful information within an image [6].
- A Graph Convolutional Network (GCN) will be used for comparison against a CNN on the same dataset as it is the closest architecture to a CNN using graphs.
- Adjacency matrices are used to express the image data and train the network where values in the matrix represent connections between nodes [2].
- A GCN encodes the input data using the following function on each layer k which then requires the network to decode the function in order to learn the features of the input data.

  -0^{th} layer: $h_v^0 = x_v$ (identity)

  $-k^{\text{th}}$ layer: $h_v^k = \sigma\left(W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1}\right)$

  Where h_v^{k-1} is the previous layer embedding of a node v, $|N(v)|$ is the number of neighbours of vertex v and the sum is to gather all neighbouring features from the previous layer $(l-1)$. W_k and B_k are trainable parameters [5].
- Activation of upcoming neurons $h_{v_i}^{(l+1)}$ are adjusted by an activation function σ where j represents neighbouring nodes of v_i, c_{ij} normalises the edge (v_i, v_j) and $W^{(l)}$ is the propagation on layer l:

$$h_{v_i}^{(l+1)} = \sigma\left(\sum_j \frac{1}{c_{ij}} h_{v_j}^{(l)} W^{(l)}\right)$$

- The loss function for a GCN is an adapted version of the loss function seen in CNNs and is summed over all vertices $v \in V$ instead of the input dataset quantity N.

PERFORMANCE COMPARISON

- The MNIST dataset will be used for classification of handwritten numbers (0-9) which features 70,000 28×28 grayscale images [1].
- Table 1 and 2 showcase the architecture of both the CNN and GNN used for training respectfully:

Table 1: CNN model layer structure (custom) using the Adam optimiser.

CNN Layers			
Layer	**Shape**	**Parameters**	**Connected Layers**
Conv2D (1)	(50, 26, 26, 6)	60	Input
AveragePooling2D (1)	(50, 13, 13, 6)	0	Conv2D (1)
Conv2D (2)	(50, 11, 11, 16)	880	AveragePooling2D (1)
AveragePooling2D (2)	(50, 5, 5, 16)	0	Conv2D (2)
Flatten	(50, 400)	0	AveragePooling2D (2)
Dense (1)	(50, 120)	48120	Flatten
Dense (2)	(50, 84)	10164	Dense (1)
Dense (3)	(50, 10)	850	Dense (2)

Table 2: GCN model layer structure using the Adam optimiser.

GNN Layers			
Layer	**Shape**	**Parameters**	**Connected Layers**
Input (1)	(None, 784, 1)	60	
Input (2)	(784, 784)	0	
GraphConv (1)	(None, 784, 32)	880	Input (1), Input (2)
GraphConv (2)	(None, 784, 32)	0	GraphConv (1), Input (2)
Flatten	(None, 25088)	0	GraphConv (2)
Dense (1)	(None, 512)	48120	Flatten
Dense (2)	(None, 10)	10164	Dense (1)

- Each model was trained on a high end workstation featuring GPUs with 48GB of RAM and both used the Adam optimiser [3] for minimising the loss function.
- The accuracy of both models on the training and validation dataset is shown in Table 3 and the loss functions which are trying to be minimised are in Figure 6:

Table 3: Accuracy and loss function results from both CNN and GCN models.

	Training Accuracy	**Validation Accuracy**	**Training Loss**	**Validation Loss**
CNN	99.81%	98.70%	0.0056	0.0650
GCN	98.06%	90.90%	0.0665	0.4201

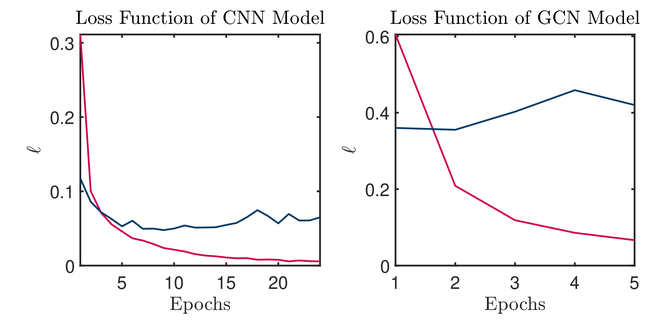

Figure 6: Loss functions from both CNN and GCN models where the pink function represents training loss and blue line represents validation loss against the epoch.

CONCLUSION & DISCUSSION

- It is clear from the loss function for the GCN model in Figure 6 (right) that the model is overfitting on the training dataset as the validation loss is stuck on a local minimum.
- The CNN model is overfitting less and performing similarly on the training and validation dataset from the accuracy in Table 3 and the similarity of plots in Figure 6 (left).
- GNNs have many benefits which include being able to use different size images in the same dataset [4], but the performance is not at a level which can outperform a CNN yet without the use of even larger datasets.
- Future work could further compare more GNN architectures to improve accuracy and overfitting results which may possibly make GNNs perform better than CNNs on MNIST.

REFERENCES

[1] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[2] Shanon Hong. An introduction to graph neural network(gnn) for analysing structured data, Mar 2020. URL `https://towardsdatascience.com/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cfdc`.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction.* MIT Press, 2022. URL `probml.ai`.

[5] Snap Stanford, Jan 2016. URL `https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/graph-neural-networks`.

[6] Varun Vasudevan, Maxime Bassenne, Md Tauhidul Islam, and Lei Xing. Image classification using graph neural network and multiscale wavelet superpixels, 2022. URL `https://arxiv.org/abs/2201.12633`.

[7] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications, 2018. URL `https://arxiv.org/abs/1812.08434`.